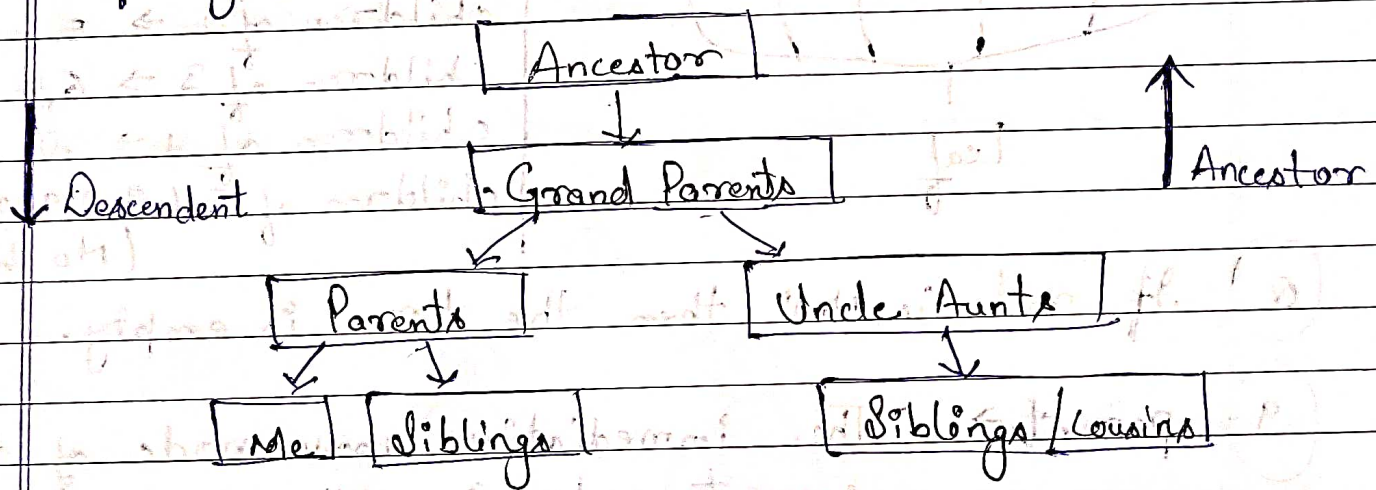
Date __/__/____ "Trees" Saathi

Till now, we have discussed all the linear data structure in which the dat is arranged in a linear manner side to side of each other. The example are array and LinkedList

We will start Non-linear

(1) Trees.

It is a data structure where data stored in a hierarchial structure for example business tree, family tree, empire tree etc.



Properties of Trees :→

1. The data is not arranged in a linear manner. It is arranged in different levels from top to bottom
2. Each element of the the tree is known as Node.
3. The node at upper level an Ancestors / Topmost Root node & lower are children of these ancestors.
4. All those node do not have any child are known as leaf nodes.

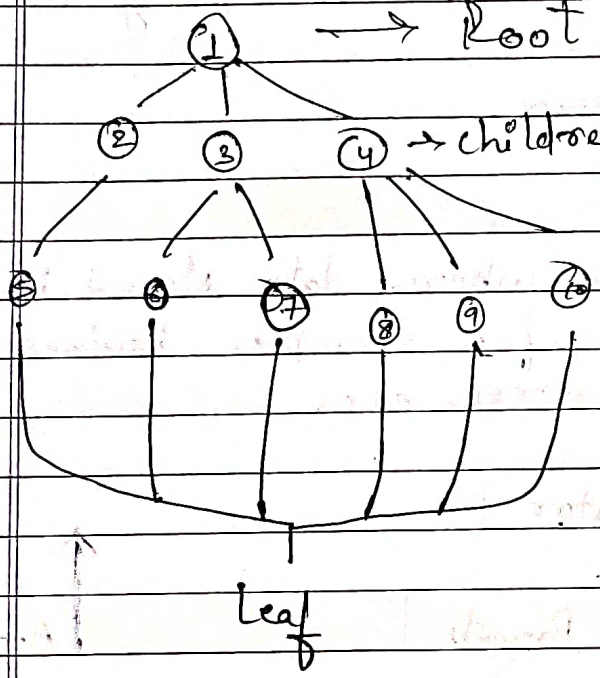
Page No.

Saathi

Date __/__/____

6 The Root node donot have any ancestors.

7 Each node can have any number of children.



→ Root → Level 1 [1]

→ children → Level 2 [2, 3, 4]

→ level 3. [5, 6, 7, 8, 9, 10]

children of 1 → 2, 3, 4
children of 2 → 5
children of 3 → 6, 7
children of 4 → 8, 9, 10
children of 5, 6, 7, 8, 9, 10 → (No child)

8 If root is Null then the tree is empty.

9 parent → The immediative above node of the current node in upper level is called. parent.

parent of 1 → No parent
parent of 2, 3, 4 → 1
etc.

10 Siblings → The nodes at the same level of the tree are called siblings. For example [2, 3, 4] are siblings [5, 6, 7, 8, 9, 10] are also siblings.

11 Ancestors → All the nodes which are at the upper level of this node are the ancestors of this node. [1, 2, 3, 4] are ancestor of [7]

Date __/__/____

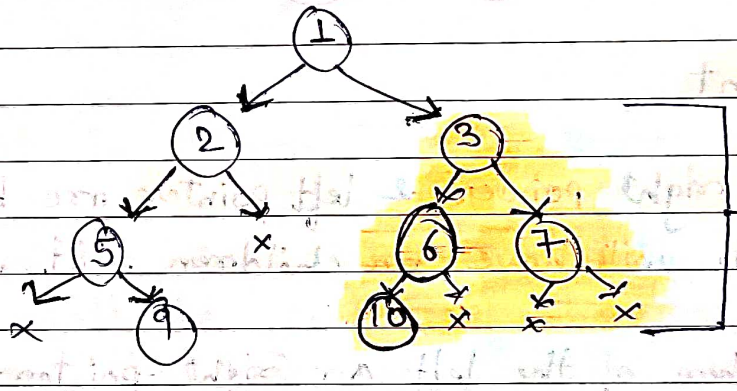
Saathi

Height of the tree → The number of levels is known as the height of the tree. In the above tree the height is '3'.

Sub-tree →

A subtree is an internal tree of an actual tree. If we want to find any subtree starting from a root node with a given data value. We take all the nodes belonging to the children of that root.

For example → Sub-tree of root value 3.




```
class Node {
    int data;
    Node* left;
    Node* Right;
vector<Node*> children; // for N-ary tree

    Node (int data) {
        this -> data = data;
        left = Right = NULL; }
```

Saathi

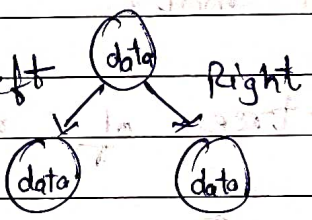
Date ___/___/_____

Binary Tree

Binary means two. A binary tree is a tree in which every node can have atmost two children.

Thus, a binary tree can have either 0, 1 or 2 children. All the properties of a tree are still valid for binary trees as well. Out of the two children, one child is called the left child and one is called the right child.

Node of a Binary Tree ?→ In previous page



* Important

1. If the right pointer & left pointer, are both NULL, then the node will have no children. It will act as leaf.

2. If either of the left or right pointer are NULL, then the node will have 1 child.

3. If no pointer is NULL, then the node will have 2 children.

Complete Binary Tree → All the leaf node are at same level

Date __/__/____

Saathi

Creation of Binary Tree (by Recursion).

```
Node* buildTree(Node* root) {

    cout << "Enter data " << endl;
    int data;
    cin >> data;
    root = new node(data);
    if (data == -1) {
        return NULL;
    }

    cout << "Left data " << endl;
    root -> left = buildTree(root -> left);
    cout << "Right data " << endl;
    root -> right = buildTree(root -> right);
}
```

Traversal in Binary Trees.

Traversal in the tree generally means to travel through all the nodes of the tree and accessing data from the tree is a crucial part of many Computer Science algorithms.

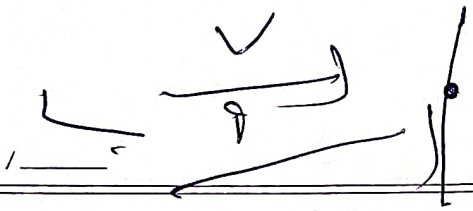
4 Traversal techniques in tree.

1. Pre Order Traversal → N L R
2. Post Order Traversal → L R N
3. In Order Traversal → L N R

(1, 2, 3) } DFS

4. Level Order Traversal. } BFS

L → Left, N → Node, R → Right



Date ___/___/_____

saathi

Pre Order Traversal → This traversal means accessing the data first before moving to its children.

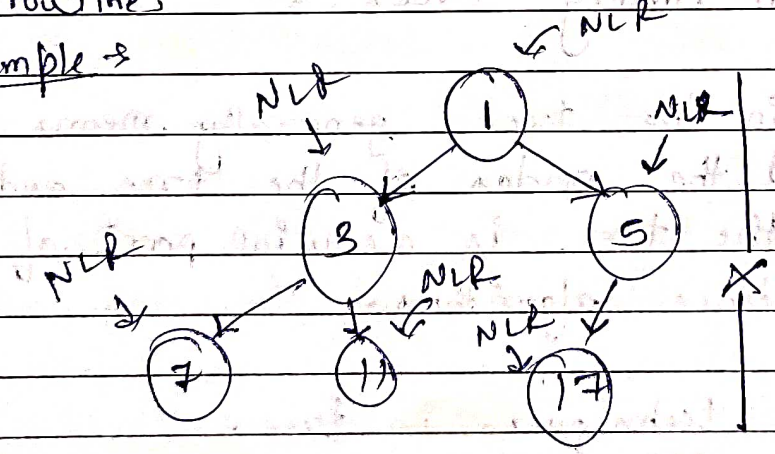
1. More formally, it means that if we are on a particular node, we access its data first and then move to the children nodes.

2. Then we move to the left node until we found any NULL node.

3. Once completed the left then we move to Right until NULL node.

It follows N L R → Node → Left → Right routine

example →



Post Order Traversal → It means accessing the data at last

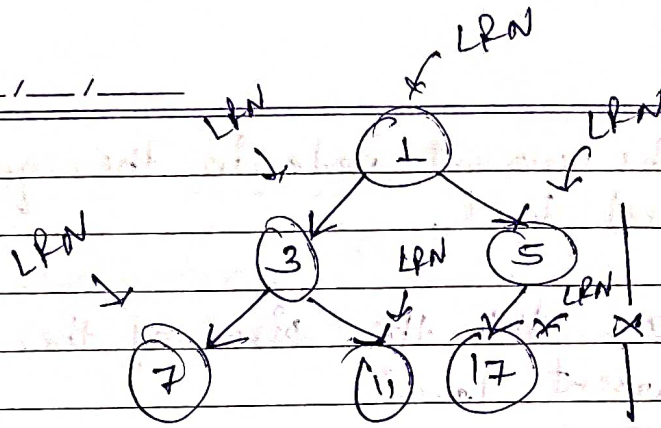
It follows the (Theory remains the same)

L R N routine

Left → Right → Node.

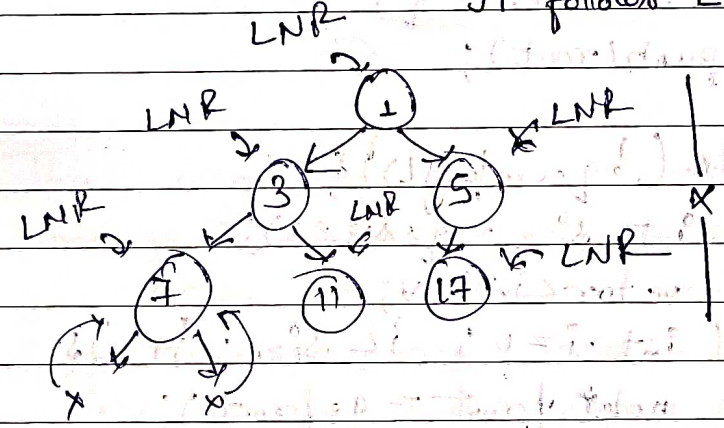
Saathi

Date __/__/____



In Order Traversal → It is a bit different but still the same method. Here we follow the access of data ~~after~~ travelling to left then data then we access the right child.

It follows LNR routine.



Level Order Traversal (1) In Level order traversal, all the nodes in the same level are shown seperately.

(2) We go level by level, that is Level 1 node will come then level 2 node will come and so on.

(3) Queue data structure is used

(4) We push the root in queue. The first level is covered. As in the queue, FIFO principle will be followed, we then push the

Page No.

children of the current node in the queue which forms the next level.

5. We traverse till the size of the queue to cover the current level.

```
vector<vector<int>> levelOrder(node* root){
    vector<vector<int>> ans;
    if(root==nullptr){
        return ans;
    }
    queue<node*> q;
    q.push(root);

    while(!q.empty()){
        int size = q.size();
        vector<int> v;
    for(int i=0; i<size; i++){
        node* front = q.front();
        v.push_back(front->data);

        if(front->left!=NULL){
            q.push(front->left);
        }
        if(front->right!=NULL){
            q.push(front->right);
        }
    }
    ans.push_back(v);
    }
    return ans;
}
```

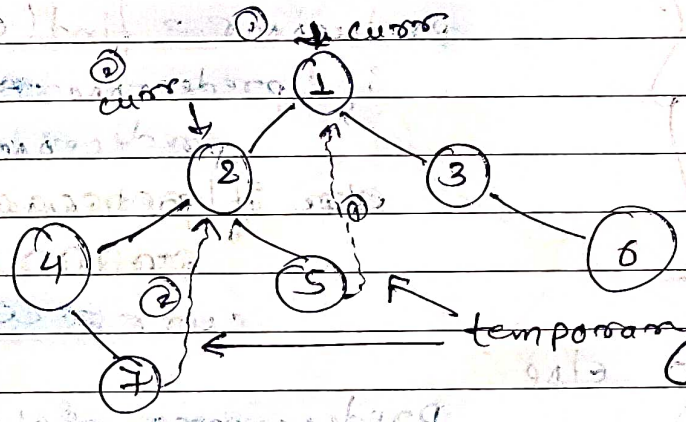
Date __/__/____

Saathi

→ Morris Traversal → In Level order, InO, PreO, PostO.
→ O(n) Time, O(n) Space.

Level order → Queue
InO, PreO, PostO → Recursion → Entire function take O(n).

But in Morris Traversal Time Complexity is O(n) but Space is O(1)



This algorithm uses temporary links to save memory because going to previous node is not possible in case of another traversal in that case we can use Morris Traversal to create temporary links to get back on a particular node after that we can delete the link.

Morris Traversal can be used we any other Traversal. It basically help to reduce Space complexity of existing Traversal.

Date ___/___/_____

Saathi

Morris Inorder Traversal

Algo:

```
curr = root
while (root != null)
    if (left not exist)
        visit (curr)  -> // Print
        curr = curr -> right
    else
        predecessor = find (curr)
        if (predecessor -> right == NULL)
            predecessor -> right = curr;
        else if (predecessor -> left == curr)
            predecessor -> left = curr;
            curr = curr -> left;
    else
        predecessor -> right == NULL
        visit ('current)
        curr = curr -> right
```

Creation of temporary Link

Deletion of temporary Link.

find predecessor ->

```
predecessor = curr - left.
while ( predecessor -> right != Null
        &&
        (predecessor -> right != curr)
```

Page No. ____

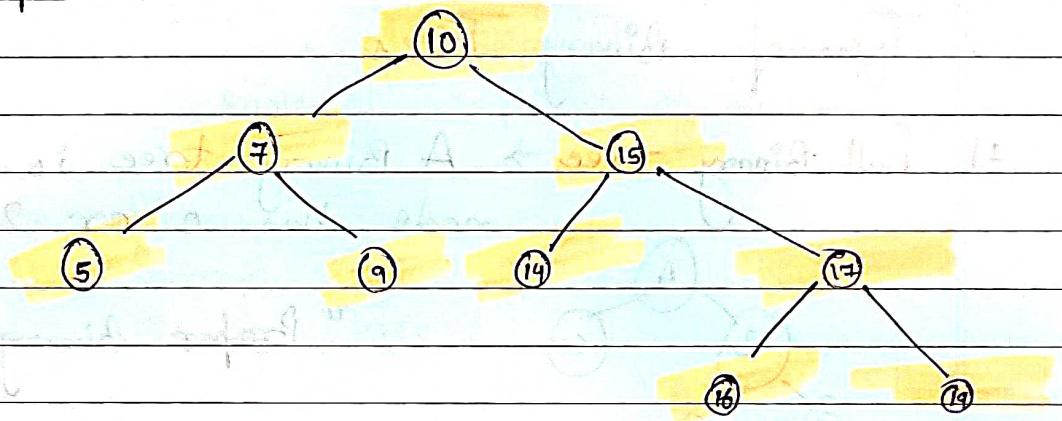
Date ___/___/_____

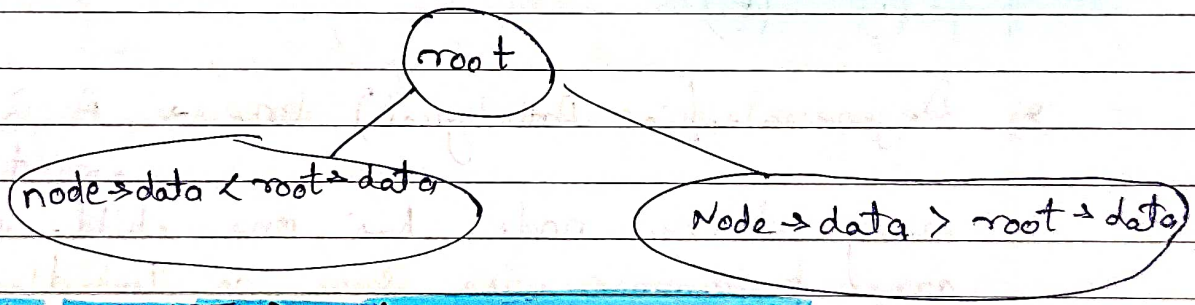
Saathi

Binary Search Tree

For every Node in BST Left part of every node is smaller than root node of that particular Node and for every node in BST right part of every node is greater than root node of that particular node.

for example.






```
Node* InsertToBST(Node* root, int d){
    if(root==NULL){
        root = new Node(d);
        return root;
    }
    if(d> root->data){
        root->right = insertToBST(root->right,d);
    }else if(d< root->data){
        root->left = insertToBST(root->left,d)
    }return root;}
```

Saathi

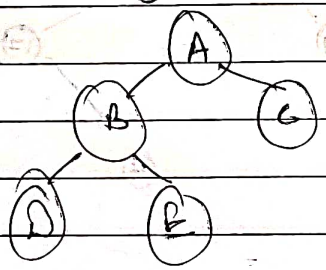
Date __/__/____

1 Important Points for a Binary Trees or Trees.
2 Types of Trees.

Definition → Binary Trees is a finite set of data item which is either empty or consists of a single item called root and two disjoint Binary tree called the left subtree & right subtree.

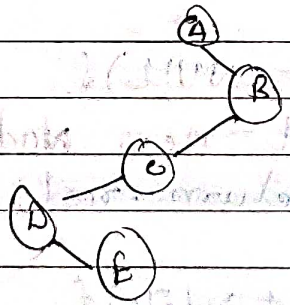
Types of Binary Trees.

1) Full Binary tree → A Binary tree is full if every node has 0 or 2 child.



"Proper Binary Tree"

2) Degenerate (or Pathological) tree → A Binary Tree or tree where every internal node has one child. Such trees are performance wise same as linked list. "single child either left or Right."



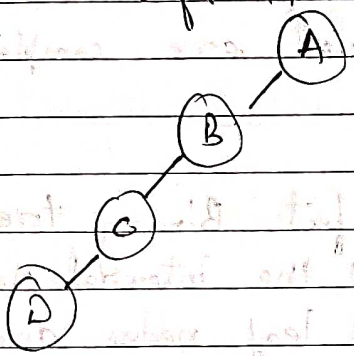
3. Skewed Binary Tree → It is a type of degenerated Binary tree in which the tree is either dominated by the left nodes or right nodes

Saathi

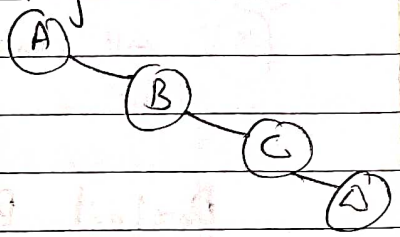
Date ___/___/_____

There are two types of Skewed Binary Tree.

Left Skewed



Right Skewed.



Types of Binary Tree on the basis of Completion of levels.

1. Complete Binary Tree.
2. Perfect Binary Tree.
3. Balanced Binary Tree.

1. A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible.

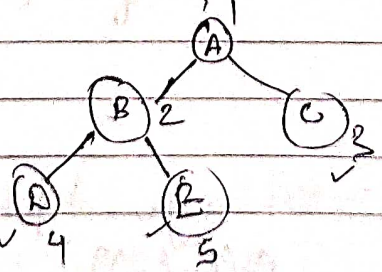
A CBT is a FBT but with two major difference.

1. Every level except the last level must be competely filled.
2. All the leaf elements must lean towards the left. The last leaf element might no have a right sibling i.e. A CBT doesn't have to be full Binary Tree.

$n = 5$

$\left(\frac{n}{2}+1\right) = 3 \rightarrow 5.$

3, 4, 5 → leaf node



$\left(\frac{n}{2}+1\right) \rightarrow n^{th}$ node is leaf node in the CBT or Heap me

Page No.

Saathi

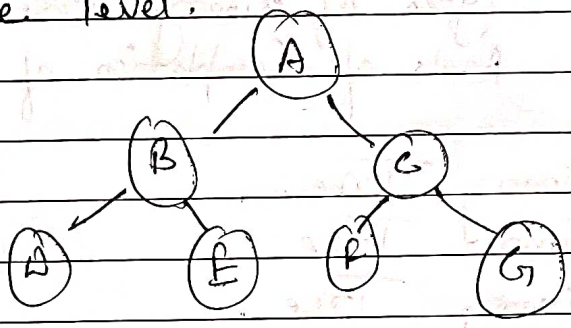
Date __/__/____

Properties → ① A CBT the number of nodes at depth d is 2^d.

② A CBT with n nodes height of the tree is $\log(n+1)$

③ All the levels except last level are completely full.

Perfect Binary Tree : A Perfect Binary tree in which all the internal nodes have two children and all leaf nodes are at the same level.



In a PBT the number of leaf nodes is the number of internal nodes plus 1.

$L = I + 1$, where L = Number of leaf nodes,
I = Number of internal nodes.

A PBT of height h (where the height of the Binary tree is the number of edges in the longest path from the root node at any leaf node in the tree, height of root node is 0) has $(2^{h+1} - 1)$ node.

A PBT of with ~~depth~~ depth of n has 2^n leaf nodes and, a total of $(2^{n+1} - 1)$ nodes.

Properties →

1. Depth of a node = Average depth of a node in PBT is $\Theta(\ln(n))$

Page No.

LC → left child.
RC → Right child.

Date __/__/____

Saathi

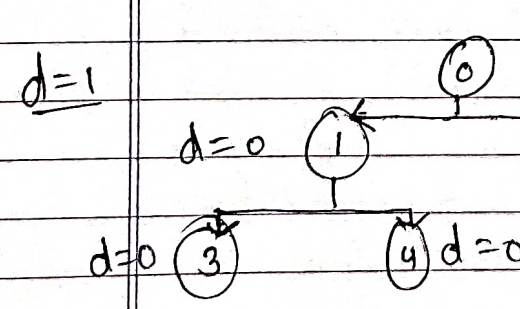
Relation between leaf nodes & non-leaf nodes →

No. of leaf nodes = no. of non-leaf nodes + 1

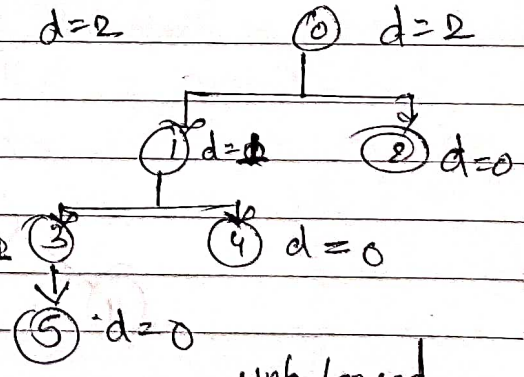
Height of the PBT → The height of a PBT with N nodes = $\log(N+1) - 1 = \Theta(\ln(n))$

3. Balanced Binary Trees → A Binary tree is Balanced if the height of the tree is $O(\log n)$ where n is the number of nodes. For example the AVL tree maintains $O(\log n)$ height by making sure that the difference between the height of the left and right subtree is at most '1'. Red-Black trees maintain $O(\log n)$ height by making sure that the number of Black nodes on every root to leaf paths is the same and that there are no adjacent red nodes. Balanced Binary Search tree are performance wise good as they provide $O(\log n)$ time for search, insert & delete.

It is a ~~type~~ type of binary tree in which the difference between the height of the left & right subtree for each node is either 0 or 1.



Balanced Binary Tree with depth at each level indicated. {Depth of node = height of LC - RC}



unbalanced BT with depth of each level indicated

In the figure, the root node having a value 0 is unbalanced with a dept of 2 units.

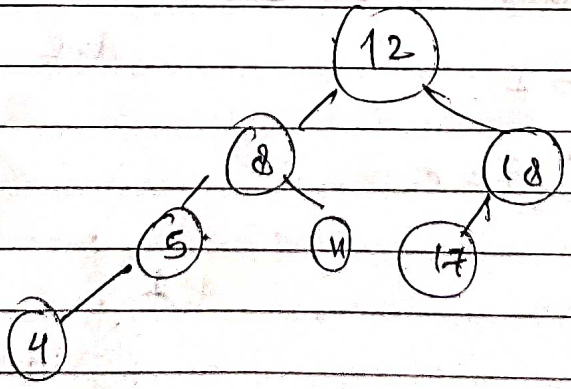
Some Special Types of Trees.

On the basis of node value the BT can be classified into.

1. BST ⟶ (Already Completed)
2. AVL
3. Red Black Tree
4. B Tree
5. B+ Tree
6. Segment Tree

(2) **AVL Tree** → It is a self balancing BST where the difference between height of left & right subtree cannot be more than one for all nodes.

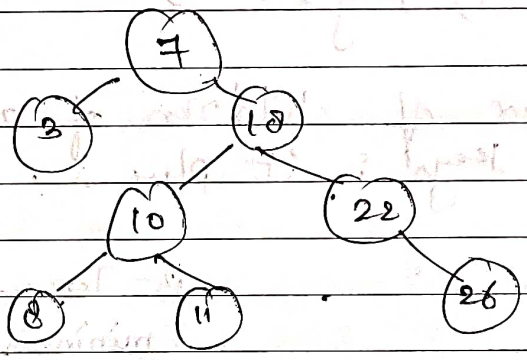
The below tree is AVL tree because the difference b/w the heights of left & right subtree for every node are less than or equal to 1.



Saathi

Date ___/___/_____

3. Red Black Tree → A RBT is a kind of self balancing binary search tree where each node has an extra bit and that bit is often interpreted as the color (red or black). These colors are used to ensure that the tree remains balanced during insertions & deletions. Although the balance of the tree is not perfect, it is good enough to reduce the searching time & maintain it around O(log n) time, where n is the total number of elements of tree. Invented in 1972 by Rudolf Bayer.



4. B-Tree → A B-tree is a type of self-balancing tree data structure that allows efficient access, insertion & deletion of data items, B-trees are commonly used in databases and file systems, where they can efficiently store and retrieve large amount of data. A B-tree is characterized by a fixed maximum degree (or order), which determines the maximum number of child nodes that a parent node can have. Each node in a B-tree can have multiple child nodes & multiple keys, and the keys are used to index and locate data items.

Page No.

Saathi

Date ___/___/_____

S/I/D in B-Tree → O(log n) n = no. of elements

Properties → (1) All the leaves are at same level.

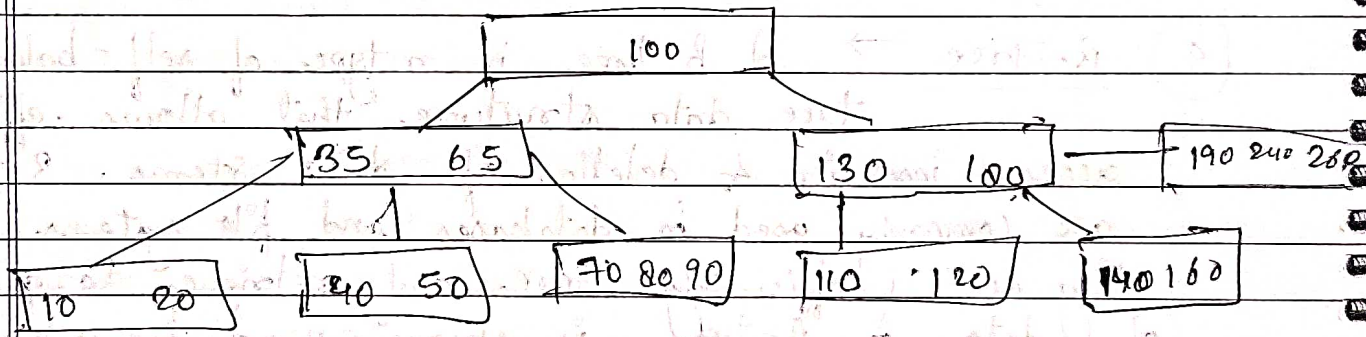
(2) B Tree is defined by the term minimum degree 't'. The value of 't' depends upon disk block size.

(3) Every node except the root must contain at least (t-1) keys.

(4) All nodes (including root) may contain at most (2*t - 1) keys.

(5) Number of children of a node is equal to the no. of keys in it plus 1.

Example → B-Tree of minimum order 5.



$h_{min} = \lceil \log_m (n+1) \rceil - 1$ → The minimum height of the B-Tree that can exist with n no. & m is the max no. of children of a node can have.

$h_{max} = \lceil \log_t \frac{n+1}{2} \rceil$ → The max height of the B-Tree can exist with n no. of nodes & t is the minimum no. of children of a non-root node can have.

& $t = m/2$

Page No.

Date ___/___/_____

Saathi

B+ Tree → It is a variation of B-Tree that is optimized for use in file systems & databases. Like a B-tree, a B+ tree also has a fixed maximum degree and allows efficient access, insertion & deletion of data items. In B+ trees, all data items are stored in the leaf nodes, while the internal nodes only contain keys for indexing & locating the data items. This design allows for faster searches & sequential access of the data items, as all the leaf nodes are linked together in a linked list.

Segmented Tree → This type of tree is also known as Statistic tree, is a tree data structure used for storing information about interval or segments. It allows querying which of the stored segments contain a given point. It is, in principle, a static structure, that is, its a structure that cannot be modified once it's built. A similar data structure is the interval tree.

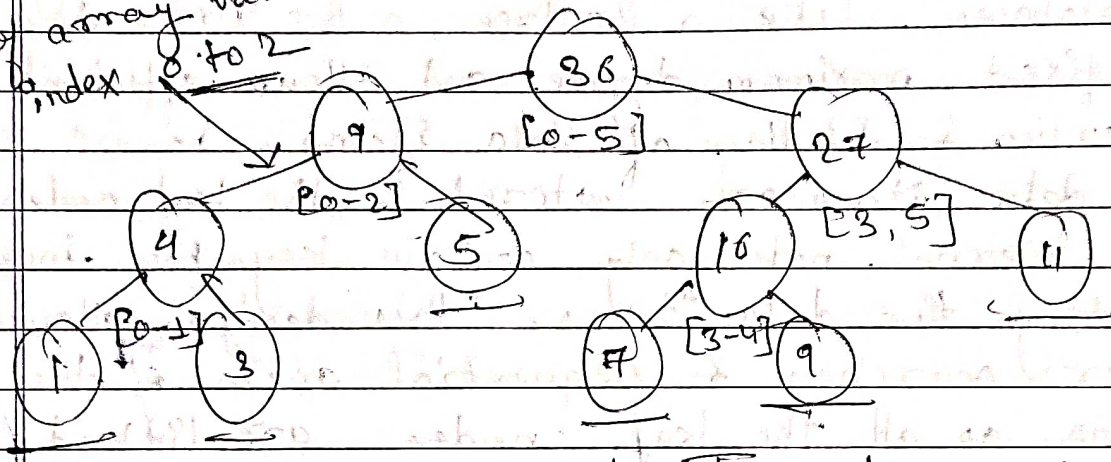
A segmented tree for a set I of n interval uses $O(n \log n)$ storage and can be built in $O(n \log n)$ time. Segment trees support searching for all the interval that contain a query point in time $O(\log n + k)$, k being the number of retrieved intervals or segments.

Page No. ____

Date ___/___/_____

Segment Tree

This node represents Sum of array values from index 0 to 2



Segment Tree for input Array {1, 3, 5, 7, 9, 11}.

Types of Operation → Add / Sub, Max / Min, GCD / LCM, AND / OR / XOR.

principle → Divide & Conquer.

R-Trees